PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F 12/04	A1	(11) International Publication Number: WO 95/02218 (43) International Publication Date: 19 January 1995 (19.01.95)

(21) International Application Number: PCT/US94/07686

(22) International Filing Date: 1 July 1994 (01.07.94)

(30) Priority Data:
08/088,788 8 July 1993 (08.07.93) US

(71) Applicant: PARK CITY GROUP, INC. [US/US]; 333 Main Street, Park City, UT 84060 (US).

(72) Inventor: BENNION, H., Richard; 9790 Sky Drive, South Jordan, UT 94095 (US).

(74) Agents: ENAYATI, Elizabeth, F. et al.; Fenwick & West, Two Palo Alto Square, Suite 500, Palo Alto, CA 94306 (US).

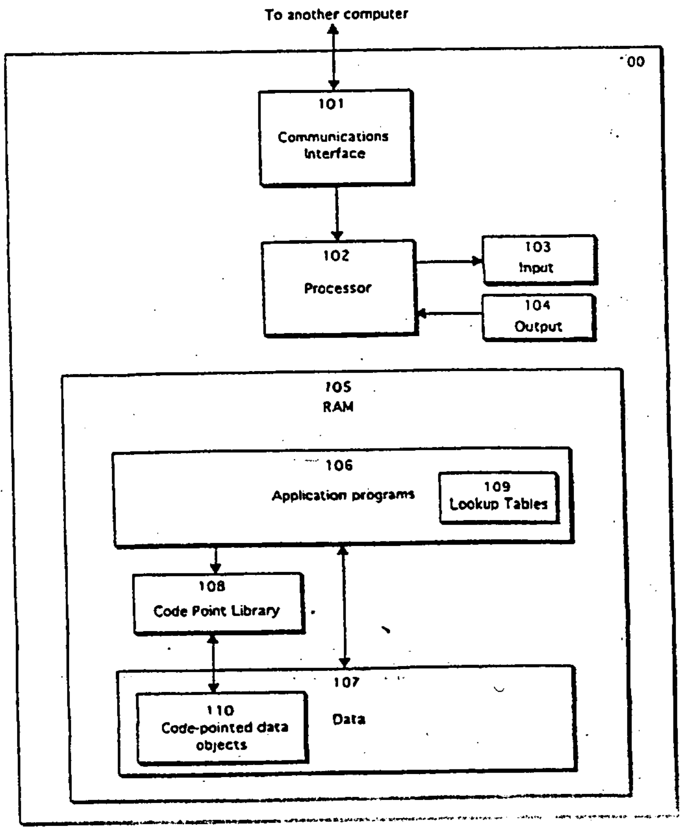
(81) Designated States: AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, ES, FI, GB, GE, HU, JP, KE, KG, KP, KR, KZ, LK, LU, LV, MD, MG, MN, MW, NL, NO, NZ, PL, PT, RO, RU, SD, SE, SI, SK, TJ, TT, UA, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: DATA MANAGEMENT USING NESTED RECORDS AND CODE POINTS

(57) Abstract

A data management system (100) and method for storing and communicating different types of data allowing variable lengths and hierarchical nesting of data records. Hierarchical structure is implicitly defined by relationships of the lengths fields of data records. The system (100) and method use data-containing records to store data and container records to contain other records in order to define the hierarchical structure and thereby greatly facilitate the movement and management of the stored data (107).



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	GB	United Kingdom	MR	Mauritania
AU	Australia	GE	Georgia	MW	Malawi
BB	Barbados	GN	Guinea	NE	Niger
BE	Belgium	GR	Greece	NL	Netherlands
BF	Burkina Faso	HU	Hungary	NO	Norway
BG	Bulgaria	IE	Ireland	NZ	New Zealand
BJ	Benin	IT	Italy	PL	Poland
BR	Brazil	JP	Japan	PT	Portugal
BY	Belarus	KE	Kenya	RO	Romania
CA	Canada	KG	Kyrgystan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo			SE	Sweden
CH	Switzerland	KR	Republic of Korea	SI	Slovenia
CI	Côte d'Ivoire	KZ	Kazakhstan	SK	Slovakia
CM	Cameroon	LI	Liechtenstein	SN	Senegal
CN	China	LK	Sri Lanka	TD	Chad
CS	Czechoslovakia	LU	Luxembourg	TG	Togo
CZ	Czech Republic	LV	Latvia	TJ	Tajikistan
DE	Germany	MC	Monaco	TT	Trinidad and Tobago
DK	Denmark	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	US	United States of America
FI	Finland	ML	Mali	UZ	Uzbekistan
FR	France	MN	Mongolia	VN	Viet Nam
GA	Gabon				

DATA MANAGEMENT USING NESTED RECORDS AND CODE POINT

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to distributed data management and, more particularly, to a persistent index storage system and method allowing storage and manipulation of data using hierarchical nested records and code points.

2. Description of the Related Art

Conventional distributed processor systems facilitate execution of several different processes simultaneously on different processors. Systems and methods for exchanging data among these processors exist in the prior art (see U.S. patent number 5,056,003 to Hammer et al. for Distributed Data Management Mechanism). Such systems and methods present techniques for reducing storage requirements by splitting a logical data stream into segments residing at different processor storage locations. Descriptor elements indicate the location and length of the various segments so that they can be reassembled into the complete logical data string.

Hammer et al. and related techniques suffer from several disadvantages. One disadvantage is that hierarchical data structures are not generally supported; if the data stream contains nested records, there is no built-in technique for navigating through the data structure. Thus, individual applications are required to deal with the hierarchical structure, as they cannot rely on the data management system to do so. This adds to programming overhead. Another disadvantage is that logical data strings have a fixed maximum length due to the fixed and limited space available for the descriptor element specifying length. Finally, existing systems generally provide only a mechanism for communicating and passing data strings among processors, but do not provide a persistent storage system.

SUMMARY OF THE INVENTION

In accordance with the present invention, there is provided a system and method of data management permitting storage and communication of different types of data, and allowing hierarchical nesting of data records and variable lengths of records. Hierarchical structure is implicitly defined by relationships of length fields of data records. Thus, movement and management of data stored according to the structure of the present invention is greatly facilitated. Individual applications are not required to deal with navigation through the hierarchical structure, since the data management system does so automatically. In addition, the data structure provides for movement of "chunks" of records that have a hierarchical relationship with one another. Finally, the system of the present invention includes format information to facilitate data transfers between applications employing different data formats.

The data structure defined by the present invention includes two types of records: data-containing records and container records. Data-containing records contain data, while container records contain other records. A code point found at the beginning of each record specifies its type. A length field facilitates variable data lengths for data-containing records, as well as implicit definition of a hierarchical structure among records. A format field facilitates movement of data among environments having differing data formats. Code point definitions may be system-wide, or application-specific, depending on the level of the particular record in the overall hierarchy.

In defining the hierarchical relationship among records implicitly by virtue of the various length fields, the data structure provides the above-recited advantages without requiring large amounts of data storage.

Brief Description of the Drawings

Figure 1 is a block diagram of a preferred hardware environment for the present invention.

Figure 2 is a diagram of the data formats used to store container records and data-containing records in the preferred embodiment.

Figure 3 is a diagram of a sample string of data bytes according to the preferred embodiment.

Figure 4 is a flowchart showing the process of deleting a record.

Figure 5 is a flowchart showing the process of adding a record.

Figure 6 is a diagram showing an example of a hierarchically-arranged collection of data fields suitable for storage according to the present invention.

Figure 7 is a diagram showing an example of a hierarchical relationship among data fields.

Figure 8 is a flowchart showing the process of inserting a record.

Description of the Preferred Embodiment

Referring now to Figure 1, there is shown a block diagram of a preferred hardware environment for the present invention. In the preferred embodiment, computer 100 may be a conventional IBM-compatible personal computer including standard communication interface hardware 101, Intel processor 102, input device 103 such as keyboard and/or mouse, output device 104 such as display screen and/or printer, and random access memory (RAM) 105. RAM 105 is divided into several sections, each of which will be described below.

Application programs 106 are stored in RAM 105 in a conventional manner. Data storage 107 stores data for use by the computer, including data objects 110 organized according to the code point structure of the present invention, as described more fully below. Some of the data may be stored on disk or other media (not shown), and may be swapped into RAM 105 as needed, according to conventional means. Code point library 108 contains the software routines for accessing data storage 107, including navigation through the data structure, interfacing, and indexing. Code point library 108 contains instructions for processor 102 for reading and modifying record lengths, and for performing operations on the stored data as described below. In response to calls from application programs 106, code point library 108 contains instructions for processor 102 to return record information and update the stored data accordingly.

Lookup tables 109 provide definitions for various code points, so that application programs 106 can interpret the data stored in data storage 107. A separate lookup table 109 may be provided for each application program 106, so that the application program 106 need only be concerned with a particular subset of the code point definitions.

Hierarchical Data

Referring now to Figure 6, there is shown an example of a hierarchically-arranged collection of data records suitable for storage according to the techniques of the present invention. For purposes of illustration, COMPANY record 600 is shown containing several records of information. All of the records contained within COMPANY record 600 relate to a single company. As can be seen, the records contained within COMPANY record 600 are arranged in a hierarchical structure.

Specifically, COMPANY record 600 contains one COMPANY DATA record 601 and two REGION records 602. COMPANY DATA record 601 describes the company as a whole, while REGION records 602 each describe a region associated with the company. Each REGION record 602 is further subdivided into a REGION DATA record 604 and two DISTRICT records 605. Each REGION DATA record 604 describes the region as a whole, while DISTRICT records 605 each describe a district associated with the region. Each DISTRICT record 605 is further subdivided into a DISTRICT DATA record 606 and two STORE DATA records 607. Each DISTRICT DATA record 606 describes the district as a whole, while STORE DATA records 607 each describe a store associated with the district.

Thus, a hierarchical structure emerges. Each individual record contained within COMPANY record 600 may be classified as either a container record or a data-containing record. Container records contain other records, while data-containing records contain data. No record is both a container record and a data-containing record. Record 600 is a container record, as are REGION records 602 and DISTRICT records 605. COMPANY DATA record 601 is a data-containing record, as are REGION DATA records 604, DISTRICT DATA records 606, and STORE DATA records 607. Thus, it can be seen that the container records serve to define and maintain the hierarchical relationship among the data-containing records.

Implicit in the hierarchical structure shown in Figure 6 is a series of nested "levels" of data. COMPANY record 600 is a Level 0 record, since no records contain it. In all code point structures according to the preferred embodiment, the first record, typically designated as Level 0, is always a container record. COMPANY DATA record 601 and REGION records 602 are Level 1 records, since they are each contained by one record. REGION DATA records 604 and DISTRICT records 605 are Level 2 records, since they are each contained by two records. DISTRICT DATA records 606 and STORE DATA records 607 are Level 3 records, since they are each contained by three records.

Referring now to Figure 7, there is shown another example of a hierarchical relationship among records. Record 700 contains several hierarchically-organized records. Container records 701 contain other records, while data-containing records 702 contain data.

Overall Data Structure

The system and method of the present invention provide techniques of efficient storage and transmission of hierarchically-organized data of the type exemplified by Figures 6 and 7.

The following table is a representation of an example of a portion of memory that is organized according to the hierarchical code point structure of the present invention. Each record in the table corresponds to one of the records of the example of Figure 6, described above.

Memory location	Record Name	Length
1-6	**Company**	**447**
7-33	Company Data	27
34-39	**Region**	**207**
40-66	Region Data	27
67-72	**District**	**87**
73-99	District Data	27
100-126	Store Data	27
127-153	Store Data	27
154-159	**District**	**87**
160-186	District Data	27
187-213	Store Data	27
214-240	Store Data	27
241-246	**Region**	**207**
247-273	Region Data	27
274-279	**District**	**87**
280-306	District Data	27
307-333	Store Data	27
334-360	Store Data	27
361-366	**District**	**87**
367-393	District Data	27
394-420	Store Data	27
421-447	Store Data	27

The above table shows the contents of a contiguous block of memory that is 447 bytes long. For purposes of illustration, the first column of the table is indented according to the level of the record. Thus, the hierarchical relationship among the records is shown.

In the table, "Memory location" indicates the location of each record within the block of memory. In the interest of simplicity for this example, all container records occupy six bytes of memory and are shown in **boldface**. All data-containing records occupy 27 bytes of memory and are shown in a normal typeface. In the preferred embodiment, container records and data-containing records may vary in length, as will be described in more detail below.

For each record, there is a "Record Name" indicating the name of the corresponding record in Figure 6. Each record also has a "Length" that indicates the total length of the record itself plus all records it contains. Thus, for example, the "Region" record that is stored at memory location 34-39 has a "Length" of 207, indicating that it contains all records residing at memory locations 34 through 240. In general, a record having a starting memory location of X and a "Length" of L contains all records residing at memory locations numbered X through X+L-1. By definition, a data-containing record has a "Length" equal to the actual length of the record itself, since a data-containing record cannot contain another record.

Thus, the placement of each record in memory, along with the "Length" values, define the hierarchical relationship among the records.

These hierarchical relationships permit rapid navigation through the data structure. If a long record is encountered that is of no interest to a particular process, the process can skip over the record (and any records nested within) by reading the "Length" and stepping forward by a number of bytes equal to the "Length". When stepping forward in this manner, the process need not read the skipped-over record into memory; only the "Length" of the first record need be read.

The hierarchical structure also permits creation of an object, or record, that contains information to be used by many different application programs 106 on the computer. Thus, for example, a level one record might represent an envelope to be handled by an electronic mail application, and might contain a number of level two records. Each level two record might be associated with a different application program 106, with the associations being defined by the code points. Each application program 106 need only be aware of the code point definitions for its associated records.

Finally, the hierarchical structure permits moving and copying of groups of nested records, or "chunks", at any level in the hierarchy. When performing an operation such as copying or moving a record, the system reads the length field and performs the operation on a continuous block of memory having the indicated length. Thus, in one operation, a record and all those records contained within it may be moved or copied.

Data Format for Records

Referring now to Figure 2, there is shown a diagram of the data formats used to store container records and data-containing records in the preferred embodiment. Container record 200 includes two fields:

- o code point 202; and
- o length 203.

Code point 202 is two bytes (16 bits) long. Three bits are used to specify the length of length field 203, which can vary from zero (indicated by 000) to four bytes (indicated by 100). Eleven bits are used to specify a user-defined code that will be interpreted by the application program 106 making use of the invention. One bit is used to indicate that this is a container record (as opposed to a data-containing record). The final bit is free.

Length 203 is from zero to four bytes long. The length of the length field 203 is determined by a portion of code point 202, as described in the previous paragraph. For container records, the length field 203 specifies the total length of all records that the current record contains (as described above). Providing variable length for this field increases the flexibility of the data format. Since the length field is up to four bytes long, the value of length 203 can range from zero to FFFFFFFF (hex).

Data-containing record 201 is similar to container record 200, except that it contains two additional fields:

- o format 204; and
- o data 205.

Format 204 is a one-byte field that specifies an attribute type that is used by the application program 106 in interpreting the stored data. Thus, the system provides for automated translation of individual records from one data format to another. Required translation operations may be provided in the code point library.

Data 205 is a variable-length field that contains the stored data.

For data-containing records, the length field 203 specifies the length of the record itself, and therefore determines the length of data field 205. As stated above, the length of a data-containing record never extends beyond the end of the record itself.

Code Points

In the preferred embodiment, code points at various levels may be associated with different application programs 106. Thus, a typical level one record might consist of a data-containing record representing an outer envelope for an electronic mail message to be processed by an electronic mail application, while a series of level two records within the level one record might consist of additional container records, each identifying a particular application program 106 to be used to process the higher-level records within. Any records of level three or greater would be passed to the individual application programs 106 for further processing.

This technique permits the same code point to be reused at various levels of the data structure. For example, a code point of 5DF at level 3 can be distinguished from a code point of 5DF at level 4, because the code points will be interpreted by different applications.

In this way, the hierarchical structure provides a mechanism for defining a hierarchical relationship among various levels of data to be used by different applications.

Other embodiments, in which the various levels of code points and records are handled differently, may also be used without departing from the essential characteristics of the invention.

Referring now to Figure 3, there is shown a sample string of 302 data bytes representing a series of hierarchically-organized records according to the present invention. Each byte is shown as a two-digit hexadecimal code 301, above which is shown the ASCII equivalent 302 for the byte. Each field is labeled to indicate its contents: "CP=CONT" is a container code point; "CP=DATA" is a data code point; "LEN" is a Length field; "FMT" is a Format field; and "DATA" is a data field. In the code points shown, the first byte is C0 for container records or 60 for data-containing records, while the second byte represents an application-defined code. Format 02 indicates ASCII data, while formats 04 and 05 indicate other types of data. Figure 3 also includes horizontal lines 303 indicating the hierarchical relationships among the records.

Operations

Given the above data structure, the following operations are available:

Delete

Referring now to Figure 4, the process of deleting a record is illustrated. To delete a record, including all the records it contains, the system rewrites the entire data structure, leaving out the records to be deleted. This is done by setting a pointer to the beginning of the data structure 402, and copying each byte of the data structure 404. Before copying each byte, the system checks 403 to see if the pointer is at a record that is to be deleted. If so, the system reads 405 the length field of the record and advances the pointer 406 by a number of bytes equal to the length of the record. After skipping forward, the system resumes copying successive bytes. Thus, the record and all the records it contains, are left out of the new data structure. When the system reaches the end of a container record 407, it reduces 408 the length field of the container record by an amount equal to the total length of all deleted records that were within the container record. The system keeps track of this total length in a stack called ICPI (Internal Code Points Information) (not shown). The system then checks 409 whether the end of the data structure was reached. If not, it repeats steps 403 through 409.

This process preserves the hierarchical relationships among surviving records when a record is deleted. For example, in Figure 6, if one of the district records 605 were to be deleted (including all records contained therein), the length field of parent region record 602 would be reduced by 87 (the length of the deleted record). In turn, the length field of parent company record 600 would be reduced by 87. The length fields of all other records would remain unchanged, as no records within those records have been deleted.

Add

Referring now to Figure 5, the process of adding a new record to the end of the data structure is illustrated. First the system sets the pointer 502 to the beginning of the data structure. Then it reads 503 the length of the first record in the structure, a level 0 record. The system then advances the pointer 504 by a number of bytes equal to the length. Thus, the pointer is positioned at the end of the data structure defined by the length of the first level 0 record. Once the pointer is at the end, the system writes 505 the new data starting at the pointer's location. As the new data is written, the system keeps track of the length of the written data, using temporary storage in the ICPI stack. After the new data is written, the length of the level 0 record is increased by the length of the newly written data 506. Thus, the level 0 record now contains the newly written data.

Although the above method was described in terms of adding data at level 1 of a data structure (so that level 0 contains the new data), the same technique could be used at any level. Once the new data is written, the system increases the value of the length records for all records containing the new data record.

Insert

Referring now to Figure 8, the process of inserting a new record into the data structure is illustrated. To insert a record, the system rewrites the entire data structure, with the new record inserted. This is done by setting a pointer to the beginning of the data structure 802, and copying each byte of the data structure

807 while successively advancing the pointer. When the pointer reaches the point at which the new record is to be inserted 803, it copies the new record 804. The code point for the new record is copied from the code point of an adjacent record having the same level. Alternatively, a new code point may be specified by the application commanding the insertion operation. When the system reaches the end of a container record 808, it increases 809 the length field of the container record by an amount equal to the total length of all records that were inserted into the container record. The system keeps track of this total length in a stack called ICPI (Internal Code Points Information) (not shown). The system then checks 810 whether the end of the data structure was reached. If not, it repeats steps 803 through 809.

This process preserves the hierarchical relationships among records when a record is inserted. For example, in Figure 6, if one of the district records 605 were to be deleted (including all records contained therein), the length field of parent region record 602 would be reduced by 87 (the length of the deleted record). In turn, the length field of parent company record 600 would be reduced by 87. The length fields of all other records would remain unchanged, as no records within those records have been deleted.

Claims

What is claimed is:

1. A data management system, comprising:

a processor;

an input device coupled to the processor, for accepting input;

an output device coupled to the processor, for generating output; and

a storage area coupled to the processor, for storing a plurality of records including a plurality of container records and a plurality of data-containing records, wherein:

each container record is hierarchically linked to at least one record and comprises a length field specifying a length; and

each data-containing record comprises:

a length field specifying a length; and

a data field containing data.

2. The data management system of claim 1, wherein:

each record is stored in a corresponding portion of the storage area; and

for each container record:

the length field defines a portion of the storage area corresponding to a subset of the records; and

the container record is hierarchically linked to the subset.

3. The data management system of claim 1, wherein each record further comprises a code point field specifying a type for the record.

4. The data management system of claim 1, wherein each data-containing record further comprises a format field specifying a data format for the data field.

5. The data management system of claim 1, wherein the plurality of records is stored sequentially in the storage area.

6. A computer-implemented method of managing data, comprising the steps of:

storing a plurality of container records, each comprising a length field specifying a length, the length field identifying a hierarchical link between the container record and at least one record; and

storing a plurality of data-containing records, each comprising a length field specifying a length and a data field containing data.

7. The computer-implemented method of managing data of claim 6, wherein the length field identifies a portion of the storage area corresponding to a subset of the records, and wherein:

the step of storing a plurality of container records includes, for each container record, storing the container record in a corresponding portion of a storage area; and

the step of storing a plurality of data-containing records includes, for each data-containing record, storing the data-containing record in a corresponding portion of the storage area.

8. The computer-implemented method of managing data of claim 6, wherein:

the step of storing a plurality of container records includes, for each container record, storing a code point field specifying a type for the container record; and

the step of storing a plurality of data-containing records includes, for each data-containing record, storing a code point field specifying a type for the data-containing record.

9. The computer-implemented method of managing data of claim 6, wherein the step of storing a plurality of data-containing records includes , for each data-containing record, storing a format field specifying a data format for the data field.

10. The computer-implemented method of managing data of claim 6, wherein the plurality of records is stored sequentially in the storage area.

11. A computer-implemented method of deleting a record in a data structure of a data management system, the data structure including a plurality of container records and a plurality of data-containing records, each record including a length field specifying a length, each data-containing record including a data field containing data, each container record having a hierarchical link to at least one record, the method comprising the steps of:

(a) selecting a record to be deleted;

(b) setting a pointer to indicate the beginning of the data structure;

(c) determining whether the pointer indicates the selected record;

(d) responsive to the pointer indicating the selected record, performing the substeps of:

(d.1) reading the length field of the selected record; and

(d.2) advancing the pointer according to the contents of the length field;

(e) responsive to the pointer not indicating the selected record, copying the record indicated by the pointer and advancing the pointer;

(f) determining whether the pointer indicates the end of a container record;

(g) responsive to the pointer indicating the end of a container record having a hierarchical link to the selected record, reducing the length field of the container record by an amount equal to the length of the selected record;

(h) determining whether the pointer indicates the end of the data structure; and

(i) responsive to the pointer not indicating the end of the data structure, repeating steps (c) through (h).

12. A computer-implemented method of adding a new record to a data structure of a data management system, the data structure including a plurality of container records and a plurality of data-containing records, each record including a length field specifying a length, each data-containing record including a data field containing data, each container record having a hierarchical linked to at least one record, the method comprising the steps of:

(a) reading a length field of a first container record;

(b) advancing a pointer according to the contents of the first container record;

(c) writing the new record at the location indicated by the pointer; and

(d) increasing the value of the length field of the first container record by an amount corresponding to the length of the new record.

13. A computer-implemented method of inserting a new record in a data structure of a data management system, the data structure including a plurality of container records and a plurality of data-containing records, each record including a length field specifying a length, each data-containing record including a data field containing data, each container record having a hierarchical link to at least one record, the method comprising the steps of:

(a) selecting an insertion point within the data structure for the new record;

(b) setting a pointer to indicate the beginning of the data structure;

(c) determining whether the pointer indicates the insertion point;

(d) responsive to the pointer indicating the insertion point, copying the new record;

(e) responsive to the pointer not indicating the insertion point, copying the record indicated by the pointer and advancing the pointer;

(f) determining whether the pointer indicates the end of a container record;

(g) responsive to the pointer indicating the end of a container record having a hierarchical link to the new record, increasing the length field of the container record by an amount equal to the length of the new record;

(h) determining whether the pointer indicates the end of the data structure; and

(i) responsive to the pointer not indicating the end of the data structure, repeating steps (c) through (i).

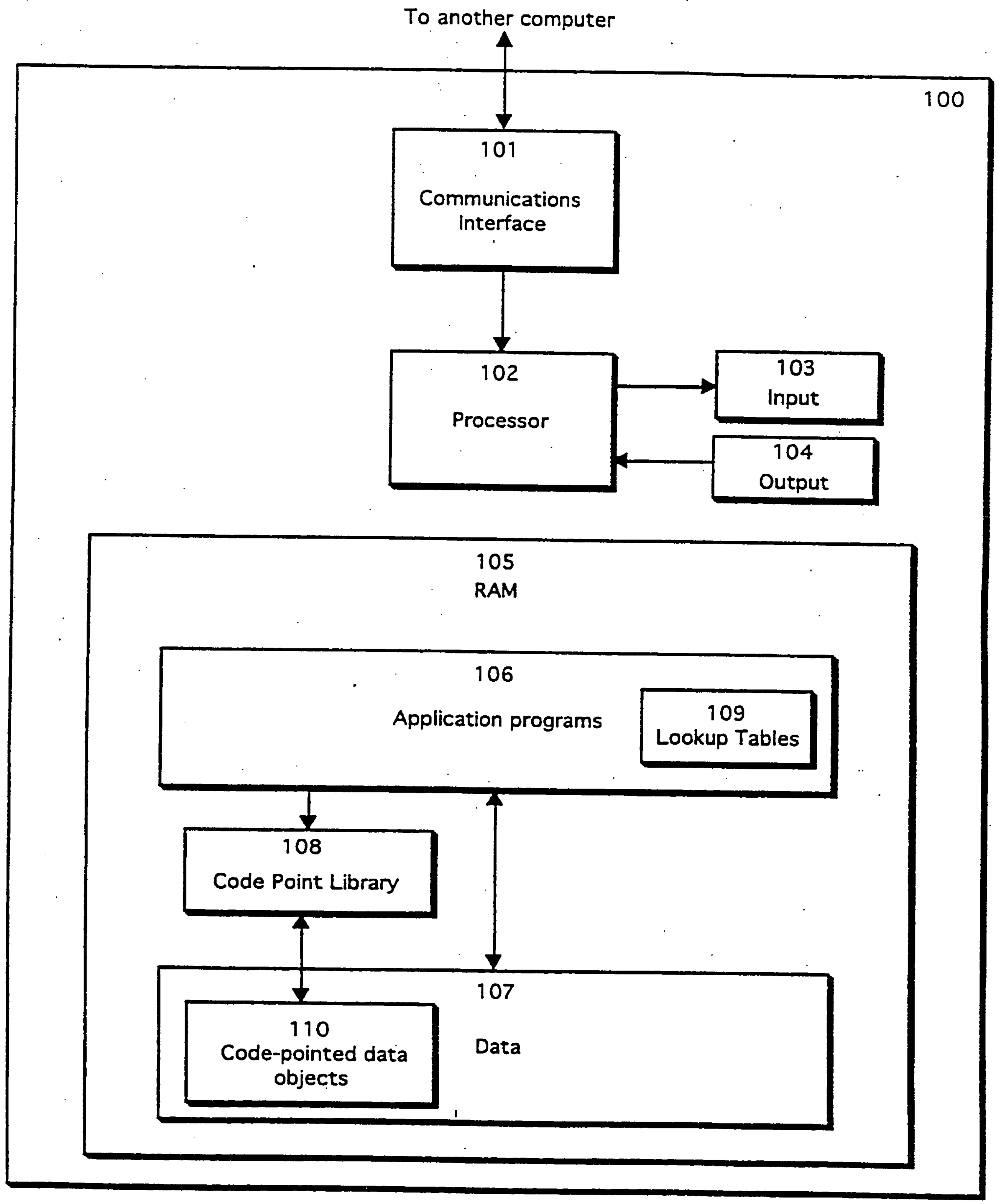

To another computer
100
101
Communications Interface
102
Processor
103
Input
104
Output
105
RAM
106
Application programs
109
Lookup Tables
108
Code Point Library
107
Data
110
Code-pointed data objects

FIGURE 1

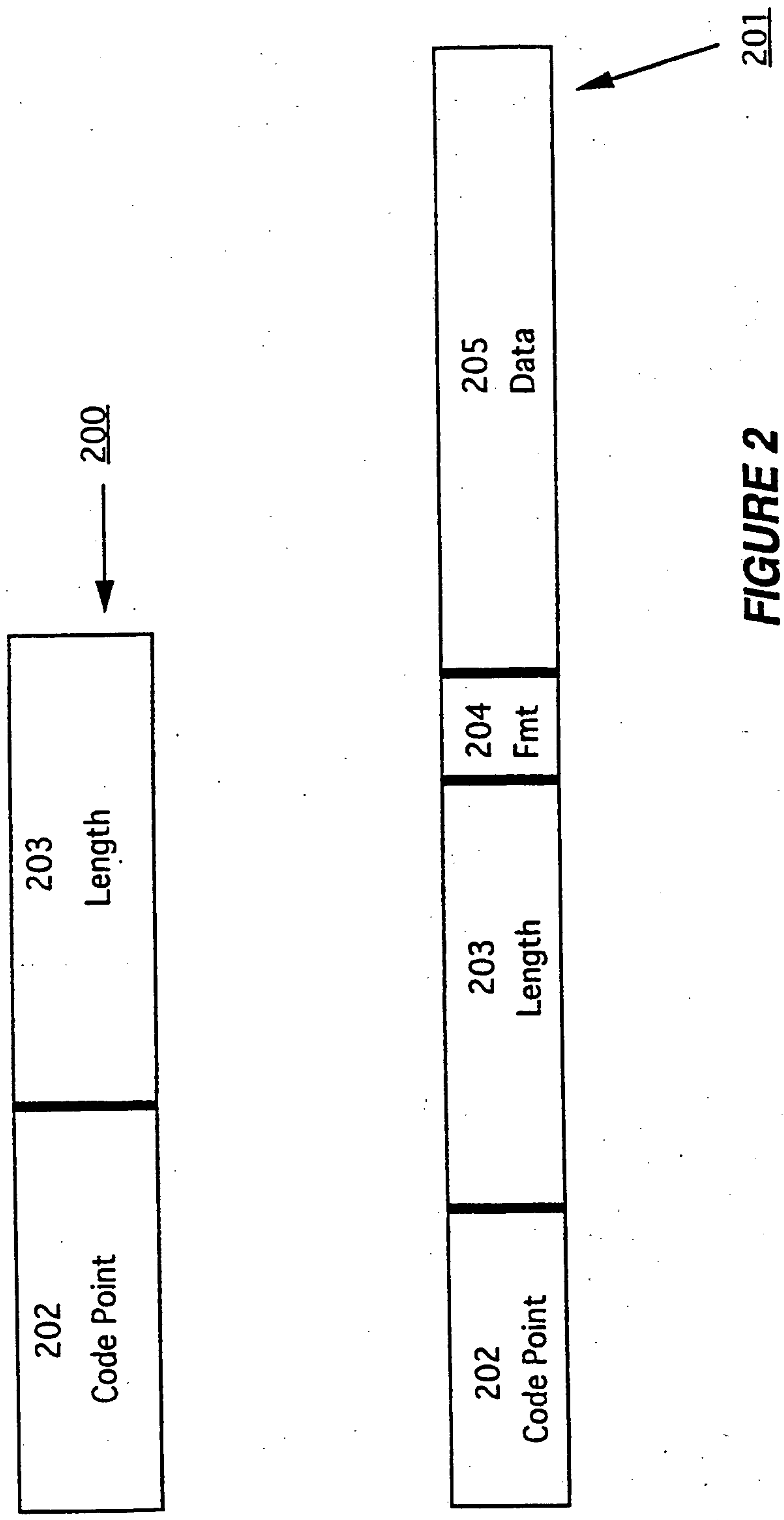
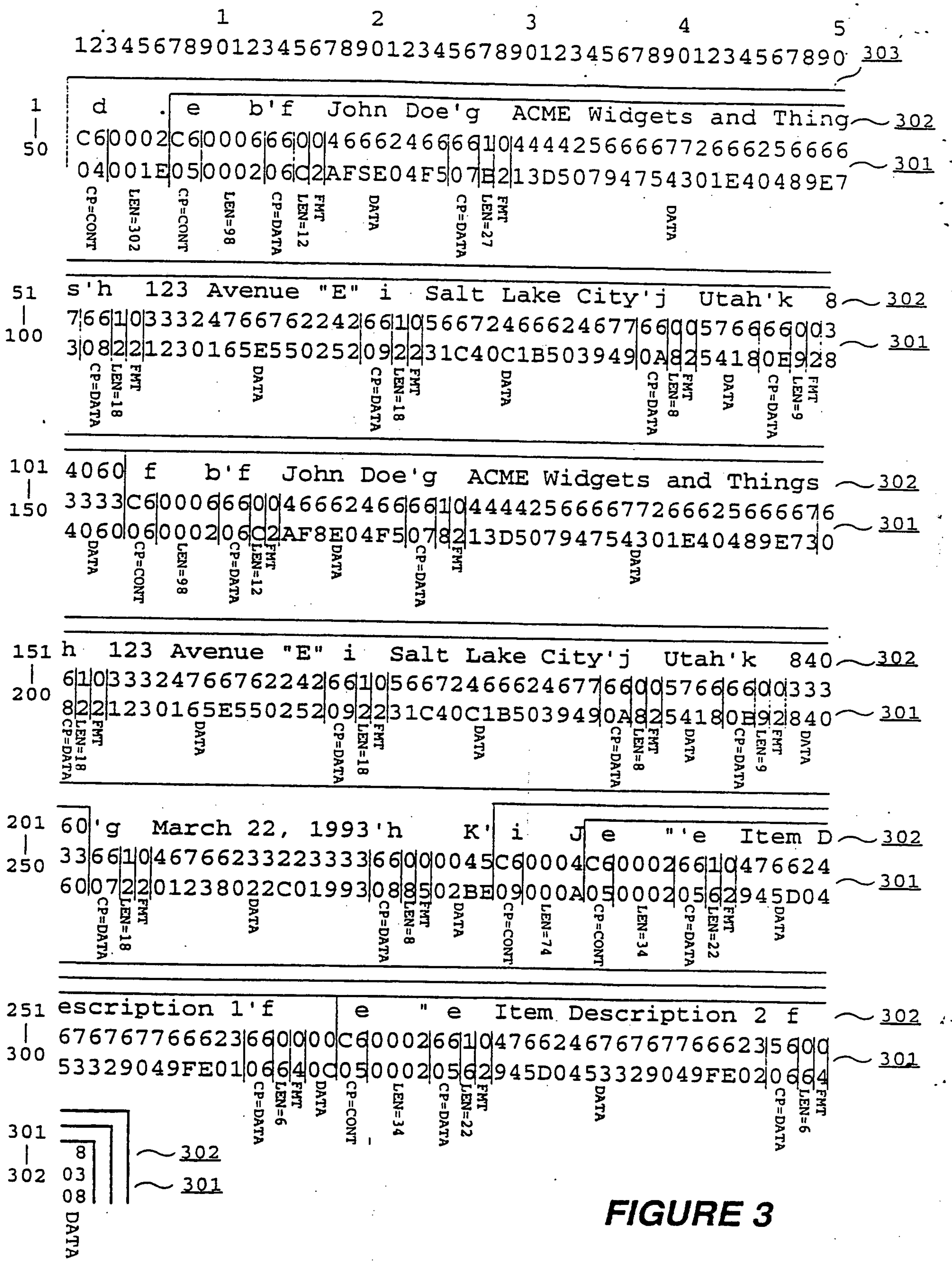

200
202 Code Point
203 Length
201
202 Code Point
203 Length
204 Fmt
205 Data

FIGURE 2


```
          1         2         3         4         5
12345678901234567890123456789012345678901234567890        303

  1   d     .e    b'f  John Doe'g  ACME Widgets and Thing    302
  |   C60002C60006660046662466661044442566667726662566666
 50   04001E0500020 6C2AF5E04F507E213D50794754301E40489E7    301

 51   s'h  123 Avenue "E" i  Salt Lake City'j  Utah'k  8     302
  |   7661033324766762242661056672466624677660057666600 3
100   3082212301 65E550252092231C40C1B5039490A825418 0E928   301

101   4060  f    b'f  John Doe'g  ACME Widgets and Things    302
  |   3333C600066600466624666610444425666677266625666676
150   40600600020 6C2AF8E04F507821 3D50794754301E40489E730   301

151   h  123 Avenue "E" i  Salt Lake City'j  Utah'k  840     302
  |   61033324766762242661056672466624677660057666600333
200   8221230165E550252092231C40C1B5039490A8254180E92840     301

201   60'g  March 22, 1993'h   K' i  J e  "'e  Item D        302
  |   33661046766233223333660000045C60004C6000266104766 24
250   6007220123802 2C01993088502BE09000A05000205629 45D04   301

251   escription 1'f    e  "  e  Item Description 2  f       302
  |   67676776662366000C6000266104766246767677666235600
300   53329049FE010664 0C050002056 2945D0453329049FE020664   301

301       8                                                  302
  |      03
302      08                                                  301
```

FIGURE 3

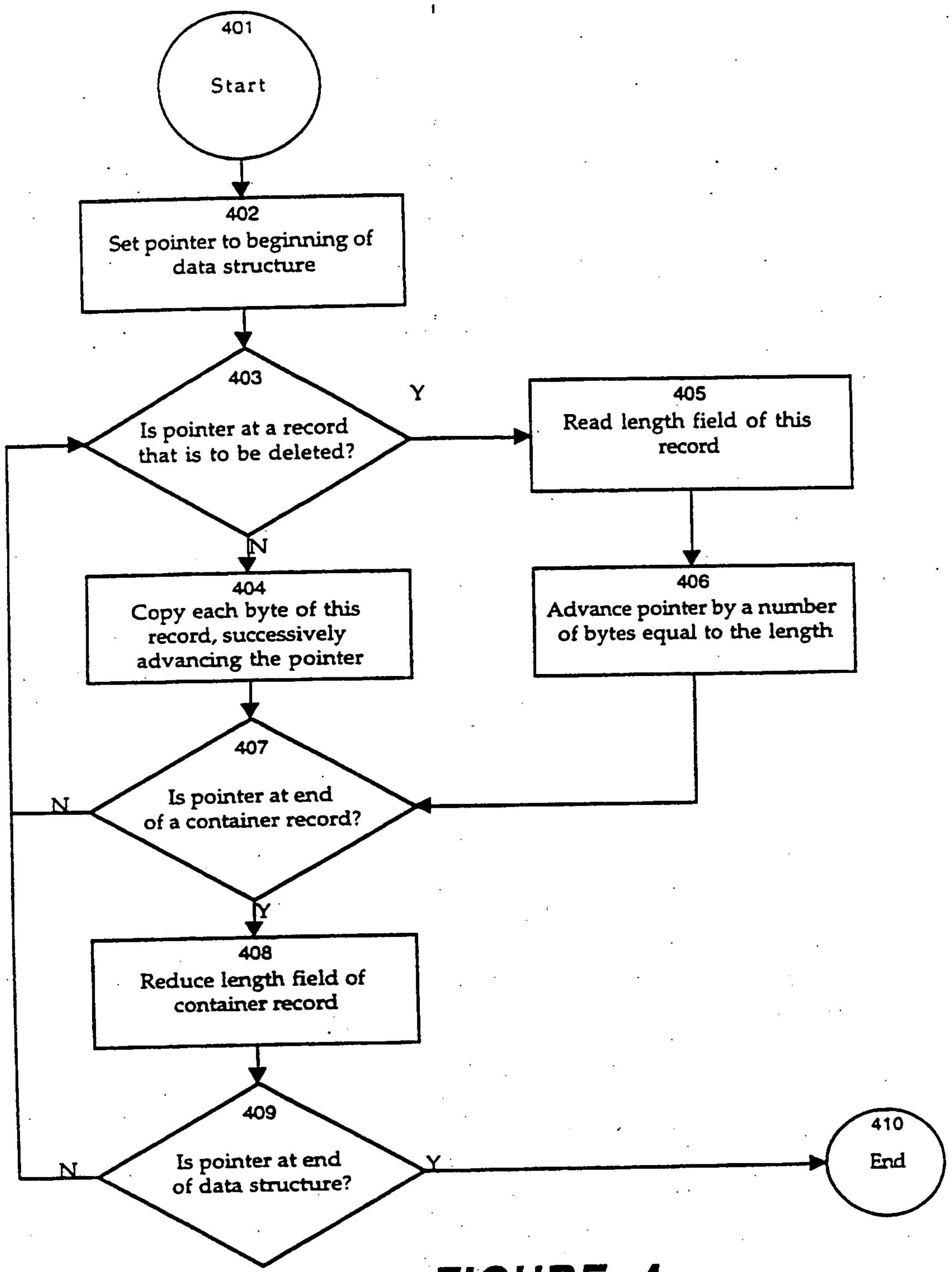

401
Start
402
Set pointer to beginning of data structure
403
Is pointer at a record that is to be deleted?
Y
N
405
Read length field of this record
404
Copy each byte of this record, successively advancing the pointer
406
Advance pointer by a number of bytes equal to the length
407
Is pointer at end of a container record?
N
Y
408
Reduce length field of container record
409
Is pointer at end of data structure?
N
Y
410
End

FIGURE 4

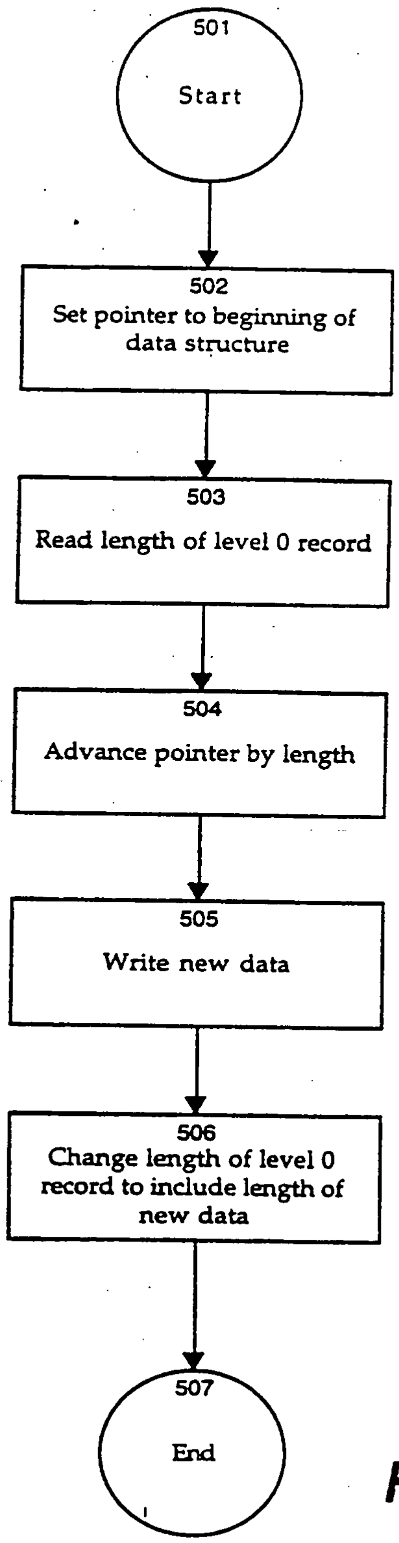

501
Start
502
Set pointer to beginning of data structure
503
Read length of level 0 record
504
Advance pointer by length
505
Write new data
506
Change length of level 0 record to include length of new data
507
End

FIGURE 5

FIGURE 6

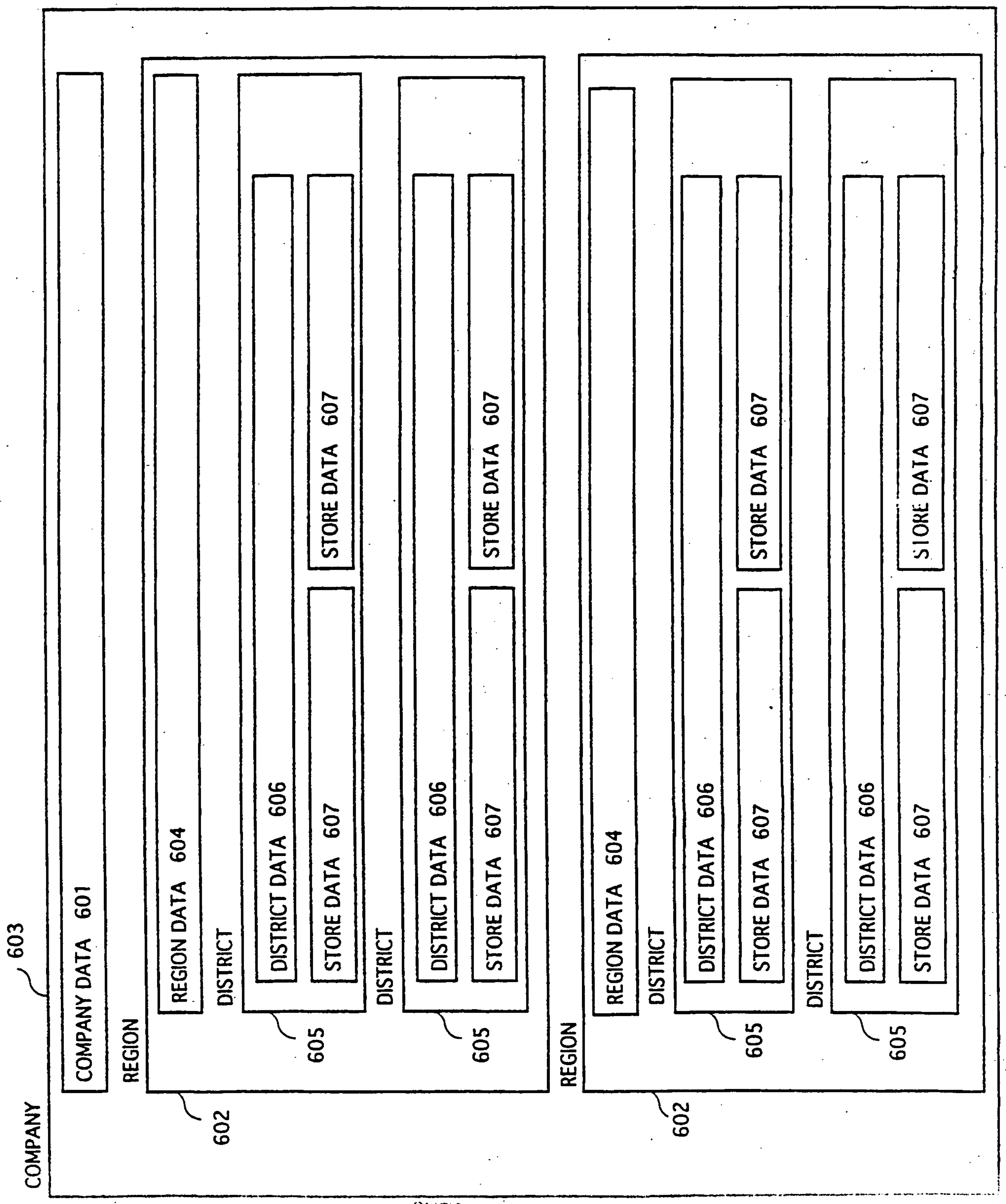
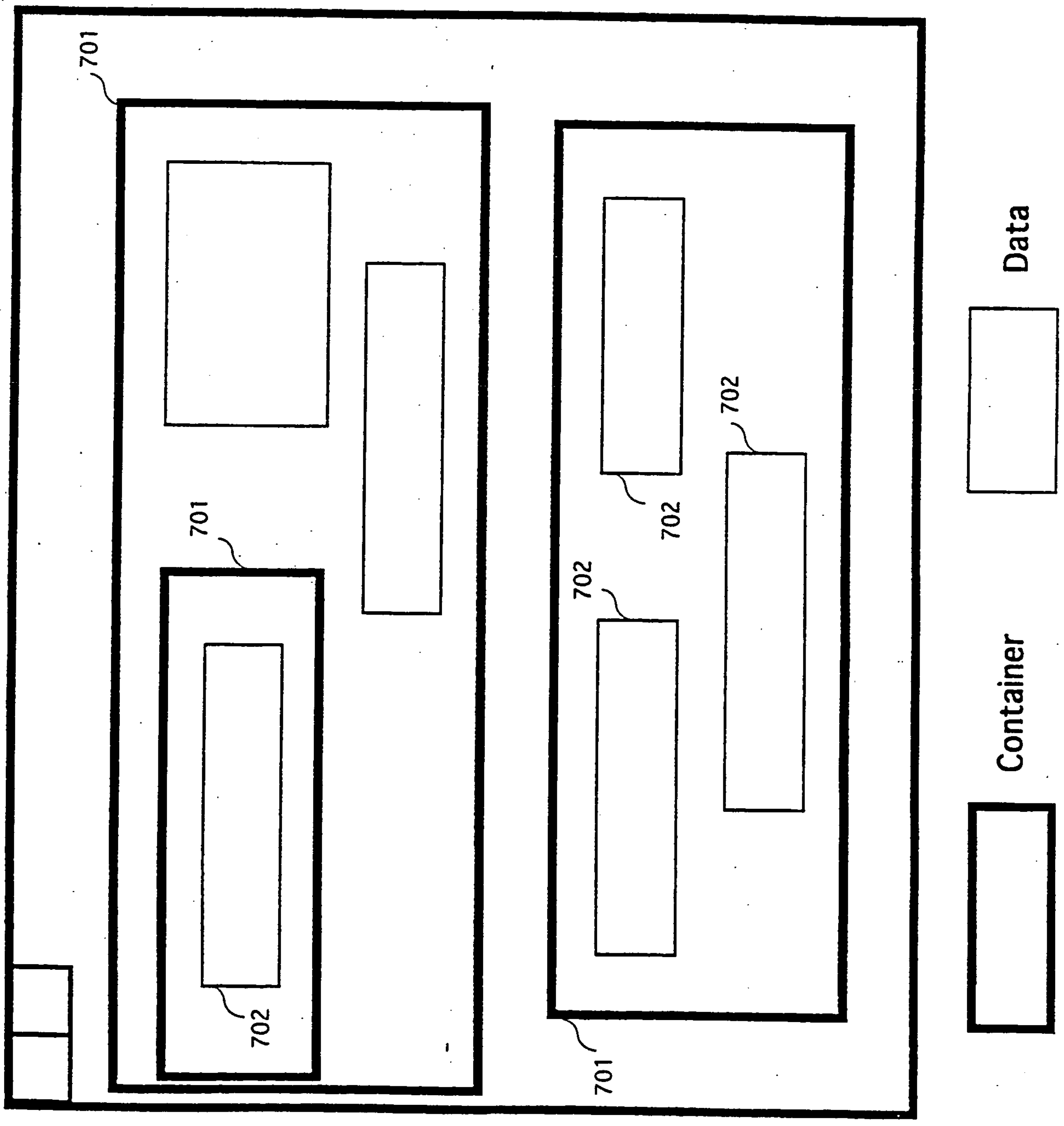

COMPANY
603
COMPANY DATA 601
REGION
602
REGION DATA 604
DISTRICT
605
DISTRICT DATA 606
STORE DATA 607
STORE DATA 607
DISTRICT
605
DISTRICT DATA 606
STORE DATA 607
STORE DATA 607
REGION
602
REGION DATA 604
DISTRICT
605
DISTRICT DATA 606
STORE DATA 607
STORE DATA 607
DISTRICT
605
DISTRICT DATA 606
STORE DATA 607
STORE DATA 607

FIGURE 7



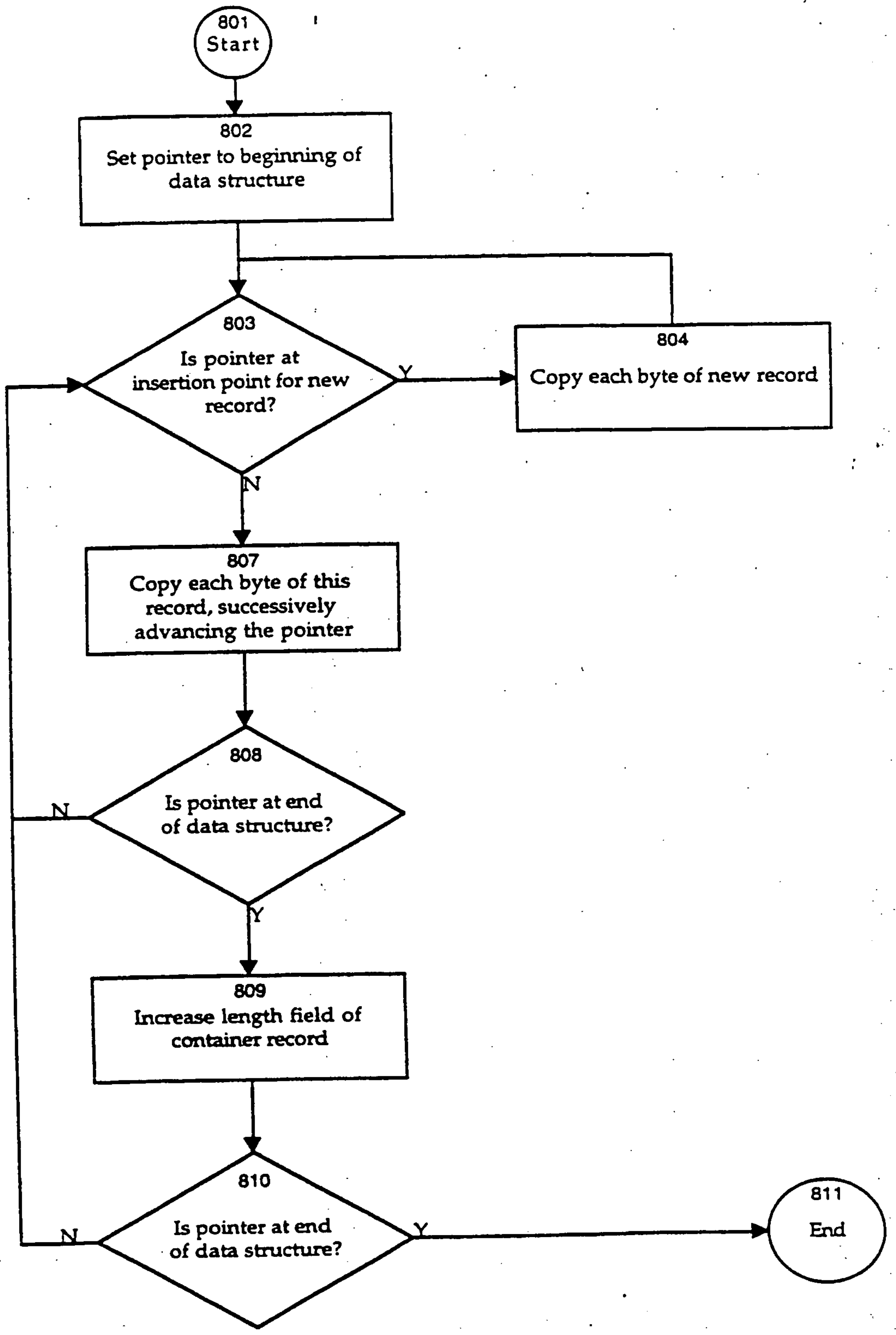

801
Start
802
Set pointer to beginning of data structure
803
Is pointer at insertion point for new record?
Y
N
804
Copy each byte of new record
807
Copy each byte of this record, successively advancing the pointer
808
Is pointer at end of data structure?
N
Y
809
Increase length field of container record
810
Is pointer at end of data structure?
N
Y
811
End

FIGURE 8

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US94/07686

A. CLASSIFICATION OF SUBJECT MATTER
IPC(5) :GO6F 12/04
US CL :Please See Extra Sheet.
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : Please See Extra Sheet.

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
APS: file systems; variable lenght record, inode, hierarchical

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US, A, 5,034,914 (Osterlund) 23 July 1991, col. 5, lines 29-61.	1-13
Y	US, A, 5,200,864 (Dunn et al.) 06 April 1993, col. 4, lines 48-68, col. 5, lines 1-66.	1-13
Y	Operating Systems: Design and Implementation, 1987, Andrew S. Tanenbaum, pages 251-273, especially page 260.	1-13

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be part of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 19 SEPTEMBER 1994	Date of mailing of the international search report OCT 13 1994
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	Authorized officer [illegible] LARRY J. ELLCESSOR

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US94/07686

A. CLASSIFICATION OF SUBJECT MATTER:
US CL :

395/600, 700

B. FIELDS SEARCHED
Minimum documentation searched
Classification System: U.S.

395/600, 700; 360/48